Document 3: Technical Specification Document

Document ID: LKP-TSD-002
Document Version: 2.0 (Dynamic AI MVP)
Status: Approved for Development
Date: 22-08-2025
Owner: Lead Developer

Document Purpose: To provide a definitive technical blueprint for the "NextLeap" MVP. This document details the system architecture, technology stack, API contract, and data flow required to build the conversational AI career advisor.

Intended Audience: Backend Developer, Frontend Developer (Streamlit), AI/Prompt Engineer.

1.0 System Architecture Overview

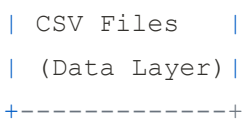
The MVP will operate on a **decoupled frontend-backend architecture** to ensure a clean separation of concerns and future scalability.

- **Frontend (Client-Side):** A **Streamlit** application serves as the User Interface (UI). Its sole responsibility is to render the conversational elements and transmit user interactions to the backend. It is a "thin client" and contains no core business or AI logic.
- **Backend (Server-Side):** A **FastAPI** (Python) application serves as the "brain" of the system. It is responsible for managing the conversation state, orchestrating calls to the LLM API, and generating the final roadmap from the static data.
- **AI Service (Third-Party):** We will use the **Google Gemini API** as our Large Language Model (LLM) to dynamically generate conversational questions.

High-Level Data Flow Diagram:

codeCode

```
+------------------+      (1. User Click: JSON)      +--------------------+
(2. Prompt)      +--------------------+
| Streamlit        | ------------------------------> | FastAPI            | -
------------------> | Google Gemini API |
| (UI Rendering)   | <------------------------------ | (Backend Logic)    |
<------------------- | (AI Service)       |
+------------------+      (5. New Question: JSON)    +--------------------+
(3. AI Response) +--------------------+
                                                            |
                                                            | (4. Filter)
                                                            |
                                                            v
                                                      +-------------+
```



2.0 Technology Stack

Component	Technology/Library	Version	Reason for Choice
Frontend	Streamlit	1.30+	Rapid UI development in Python, perfect for creating interactive, data-focused prototypes.
Backend	FastAPI	0.100+	High-performance Python framework for building robust APIs. Asynchronous support is ideal for handling external API calls.
AI Model	Google Gemini API	Pro	Advanced reasoning, excellent Hinglish support, and a generous free tier for development.
Data Handling	Pandas	2.0+	Industry standard for reading and manipulating CSV data efficiently.
API Testing	Postman / FastAPI Docs	-	For rigorously testing the backend API endpoints independently of the frontend.
Deployment	Streamlit Community Cloud (Frontend), Render/PythonAnywhere (Backend - Free Tier)	-	Easy, scalable, and free deployment options suitable for a hackathon.

3.0 API Contract Definition

The entire communication between the frontend and backend is managed through a single, powerful API endpoint.

- **Endpoint:** `/next-step`
- **Method:** `POST`

- **Description:** This endpoint orchestrates the entire user journey. The frontend sends the complete current state of the conversation, and the backend returns the next UI state to be displayed.

A JSON object containing the complete user session state.

codeJSON

```
{
  "session_id": "unique_user_session_string",
  "current_page": "quiz_1", // The page the user is currently on
  "conversation_history": [
    {"role": "user", "parts": ["After 10th"]},
    {"role": "model", "parts": ["{\"type\":\"question\",...}"]}
  ],
  "user_choices": {
      "stage": "after_10th",
      "q1_answer_profile": "Analytical",
      "happiness_score": null
  }
}
```

The backend will always respond with a JSON object that tells the frontend exactly what to render next.

1. Next Question Response:

codeJSON

```
{
  "next_page": "quiz_2", // Tells Streamlit which page/state to render next
  "ui_elements": {
    "type": "question",
    "question_text": "Interesting! What about Science excites you most?",
    "options": ["The physical world", "Living things"]
  }
}
```

2. Happiness Slider Response:

codeJSON

```
{
  "next_page": "happiness_check",
  "ui_elements": {
    "type": "slider",
```

```
    "logical_recommendation": "Engineering",
    "question_text": "How excited does this path make you feel?"
  }
}
```

3. Final Roadmap Response:

codeJSON

```
{
  "next_page": "roadmap",
  "ui_elements": {
    "type": "roadmap",
    "career_info": { "name": "Engineering (CSE)", "description": "...",
"skills": ["Python", "..."] },
    "college_info": [ { "name": "IIT Bombay", "city": "Mumbai" }, ... ],
    "checklist_data": [ { "phase": "Foundation", "tasks": ["..."] }, ... ]
  }
}
```

4.0 Backend Logic Flow

When the `/next-step` endpoint is called, the backend will:

1. **Parse Request:** Extract `current_page`, `conversation_history`, and `user_choices`.
2. **Determine Next Action:** Based on the `current_page` and `user_choices`, decide the next step (e.g., call LLM, check happiness score, or generate roadmap).
3. **Execute Action:**
    - **If needs new question:** Construct the Master Prompt with the conversation history and call the Gemini API.
    - **If needs final roadmap:** Analyze the `user_choices` (including happiness score), filter the CSV files using Pandas, and structure the final roadmap data.
4. **Construct Response:** Build the appropriate JSON response object (Next Question, Happiness Slider, or Final Roadmap).
5. **Send Response:** Return the JSON to the Streamlit frontend.
6. **Error Handling:** Implement robust try-except blocks, especially for the external API call to Gemini, to handle potential failures gracefully.